ATTORNEY DOCKET NO. 06502-0260

UNITED STATES PATENT APPLICATION

OF

BRIAN T. MURPHY

AND

ROBERT W. SCHEIFLER

FOR

LOOKUP DISCOVERY SERVICE

Related Applications

The following applications are relied upon and are hereby incorporated by reference in this application.

U.S. Provisional Application No. 60/138,680, entitled "Jini™ Technology Helper Utilities and Services," bearing attorney docket no. 06502.6010-00000.

U.S. Patent Application No. 09/044,931, entitled "Dynamic Lookup Service in a Distributed System," bearing attorney docket no. 06502.0110-00000. P.N. 6185611

U.S. Patent Application No. 09/044,939, entitled "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System," bearing attorney docket no. 06502.0112-00000. allowed not issued

U.S. Patent Application No. 09/030,840, entitled "Method and Apparatus for Dynamic Distributed Computing Over a Network," and filed on February 26, 1998. 6496070

Field of the Invention

The present invention relates to a system and method for communicating with a device in a distributed system and more particularly to a system and method for utilizing a third party server to facilitate communication between devices on a network.

Background of the Invention

Before devices can communicate and share resources with each other in a modern distributed computing environment, each remote terminal or client, must have a reference identifier for the target device (printer, mass storage device, telephony interface, etc.). The reference identifier essentially contains sufficient addressing information to direct the client to the target device. In essence, the other devices on the distributed network must have access to code, device drivers, and interface port information for any other device that joins the network. In one conventional distributed system, services are logically grouped in a distributed Jini™ system, referred to as a "Djinn" as described in copending U.S. Patent Application Serial No. 09/044,931, entitled "Dynamic Lookup Service in a

now U.S. Patent 6,185,611 issued Feb. 6, 2001

Distributed System," assigned to a common assignee, which has been previously incorporated by reference. A "service" refers to a resource, data, or functionality that can be accessed by a user, program, device, or another service and that can be computational, storage related, communication related, or helpful in providing access to another user. Examples of services provided as part of a Djinn include devices, such as printers, displays, and disks; software, such as applications or utilities; information, such as databases and files; and users of the system. When joining a Djinn, the user or device adds zero or more services to the Djinn and may access, subject to security constraints, any one of the services it contains. Thus, devices and users federate into a Djinn to share access to its services. The services of the Djinn appear programmatically as objects of the Java™ programming environment, which may include other objects, software components written in different programming languages, or hardware devices.

The goal of any well-behaved Jini service, implemented within the bounds defined by the Jini technology programming model, is to advertise the service it provides in at least one Jini Lookup service. Such a process is known as registering with, or joining the lookup. To demonstrate this so-called good behavior, a service complies with both the multicast discovery protocol (group discovery) and the unicast discovery protocol (locator discovery) in order to discover the lookup services it is interested in joining. In brief, a service wishing to register in a lookup service first sends a multicast message on the network, which is responded to by a discovery server. Each discovery server receiving the message responds with a reference to an associated lookup service. After receiving this response, the service may register itself in the lookup service by sending a unicast message to the lookup service. Once registered, the service receives a lease object from the lookup service indicating that that service will be registered for a particular period of time. It is the responsibility of the service to renew this lease before it expires, otherwise its registration will be deleted. Leases are further described in co-pending U.S. Patent Application Serial No. 09/044,923, now U.S. Patent 6,263,350 issued July 17, 2001, entitled "Method and System for Leasing Storage," assigned to a common assignee, which has been previously incorporated by reference.

In order to efficiently utilize existing memory space and to conserve processing power, a service may be implemented as an activatable object. That is, the service may be selectively activated (stored in memory) or deactivated (removed from memory), depending upon whether it is currently

being used by another network service. Both the service and the lookup service are activatable objects, thus posing a problem. Specifically, for a service to register with a lookup service, both the service and the lookup service must be active and operational, respectively. If not, the service either will not be looking for the lookup service because it is inactive, or it will not be able to find the lookup service because the lookup service has crashed. Based on these problems, it is desirable to improve the process and apparatus for performing lookup service registration.

Summary of the Invention

Methods and systems consistent with the present invention provide a lookup discovery service that solves the problem encountered by conventional systems. Specifically, on behalf of client or service entities, the lookup discovery service locates lookup services that the entities cannot locate themselves; either because the lookup service is out of reach, or the entity is inactive. In providing a service with access to lookup services beyond its reach, the lookup discovery service enables a service to register with lookup services outside of its multicast radius. While multicast radii can and often do overlap, a multicast broadcast transmitted by an entity will only reach those lookup services within the entity's multicast radius. Referring to FIG.1, it is shown that multicast radius 80 is comprised of client 50, lookup services 71 and 72, and lookup discovery service 95. Multicast radius 85 is comprised of lookup discovery service 95 and lookup services 73, 74 and 75. A multicast broadcast transmitted by client 50 will only reach those entities located inside of multicast radius 80. However, as a result of its location, multicast broadcasts by lookup discovery service 95 will reach all entities located inside multicast radius 80 as well as those located inside multicast radius 85. Link 60 indicates that client 50 has discovered lookup service 71. Link 90 indicates that lookup discovery service 95 has discovered lookup services 71, 73 and 74 and that client 50 has acquired a reference to lookup discovery service 95 through lookup service 71. Links connecting lookup services 72 and 75 with either client 50 or lookup discovery service 95, were omitted simply to illustrate the fact that clients, lookup discovery services and lookup services located inside the same multicast radius may, but do not necessarily have to discover each other. As will be shown later, an entity that has not been discovered can not access resources located in another entity.

If client 50 were provided with references to lookup services located outside of its multicast radius 80 that contain services needed by the client, it could contact each lookup service and retrieve the desired service references despite the fact that such services aren't available inside its multicast radius 80. One way to achieve this might be to configure the client 50 to find and access lookup discovery service 95 which would employ the multicast discovery protocol to discover nearby lookup services belonging to groups in which the client has expressed interest. After acquiring references to the targeted lookup services, the lookup discovery service 95 would pass those references back to client 50, providing it with access to each lookup service. In this way, client 50 participates in the multicast discovery protocol through a proxy relationship with the lookup discovery service 95, gaining access not only to each discovered lookup services 71-75 but also to all of their registered services.

Systems consistent with the present invention provide a lookup discovery service that continuously monitors associated lookup services and provides information about lookup services of interest to registered clients. Lookup discovery services in effect provide the capability for clients to maintain a more complete representation of available lookup services and in turn, network services. As a result, clients gain access to a more comprehensive collection of services than with prior art systems. Lookup discovery services will further provide clients with first time, automated access to network services located outside of the client's multicast radius. Thus, if a service is not registered with a lookup service that is inside of the client's multicast radius, the client can query a lookup discovery service to identify lookup services, located elsewhere on the network, with which the service of interest may be registered. Additionally, the lookup discovery service provides an event mechanism with notification semantics that allows the client to be notified of the arrival of new lookup services of interest as well as any changes in the state of the current lookup services on the network.

In order for the lookup discovery service to be able to link an entity with lookup services in the way described above, the lookup discovery service must be registered with a lookup service having a location that is either known to the entity, or within the multicast radius of the entity. Furthermore, the lookup discovery service must be running on a host that is located within the multicast radius of the lookup services with which the entity wishes to be linked. That is, the entity

must be able to find the lookup discovery service; and the lookup discovery service must be able to find the other desired lookup services. Referring again to FIG. 1, it is shown that lookup discovery service 95 has discovered lookup services 71, 73 and 74, and that the client 50 is registered with the lookup discovery service 95. Client 50, utilizing the functionality of lookup discovery service 95 can access lookup services 71, 73 and 74. Client 50, however, may not access lookup services 72 or 75 until those services are first discovered by lookup discovery service 95 and their references are passed to the client 50. Note that since client 50 and lookup service 72 are inside the same multicast radius 80, client 50 may independently discover lookup service 72 utilizing the process disclosed in previously referenced application entitled, "Apparatus and Method for Providing Downloadable Code for Use in Communicating with a Device in a Distributed System." Without the assistance of lookup discovery service 95, client 50 could only discover lookup service 75 via unicast discovery. In other words, unless lookup discovery service 95 discovers lookup service 75, client 50 could only access that lookup service if it was otherwise in possession of the lookup service's unique locator information.

To address scenarios such as those described above, the lookup discovery service participates in both the multicast discovery protocol and the unicast discovery protocol on behalf of a registered discovering entity or client. The lookup discovery service will listen for and process multicast announcement packets from Jini Lookup services, and will, until successful, repeatedly attempt to discover specific lookup services that the client is interested in accessing.

Upon discovery of an as yet undiscovered lookup service of interest, the lookup discovery service notifies all entities that have requested the discovery of that lookup service, that such an event has occurred. The event mechanism employed by the lookup discovery service satisfies the requirements defined in the previously referenced, co-pending U.S. Application No. 09/044,790 entitled "Method and Apparatus for Determining Status of Remote Objects in a Distributed System," which has been incorporated by reference. Once a client is notified of the discovery of a lookup service, it is left to the client to define the semantics of how it interacts with that lookup service. For example, the client entity may wish to join the lookup service or simply query it for other useful services (or both).

In accordance with methods consistent with the present invention, a method is provided in a data processing system having a lookup discovery service for accessing associated lookup services. This method receives a client request by the lookup discovery service to identify lookup services and returns the location of a lookup service of interest.

In accordance with methods consistent with the present invention, another method is provided in a data processing system having a client computer, a lookup discovery service for accessing associated lookup services, and a lookup service containing service stubs for accessing associated network services. This method transmits a request from the client to the lookup discovery service, identifying one of the associated lookup services to be accessed. Then through participation in the discovery process, the lookup discovery service retrieves the locator information for the lookup service and returns that locator information to the client.

In accordance with systems consistent with the present invention, an apparatus is provided that comprises a lookup service, a lookup discovery service having access information for a plurality of lookup services, and a client having a program that sends out requests for lookup services to an associated lookup discovery service that accesses associated lookup services and returns lookup service locator information that facilitates access by the client to the identified lookup service.

In accordance with yet another aspect of the present invention, as embodied and broadly described herein, a computer program product comprises a computer usable medium having computer readable code embodied therein for controlling a data processing system having a lookup service with associated network services available in a distributed system, a lookup discovery service having locator IDs to a plurality of lookup services, and a client. The code comprises the steps of sending a request to identify locator information for a desired lookup service to a lookup discovery service; searching lookup services associated with the lookup discovery service to identify a lookup service that satisfies the client's request; and returning the locator information of the lookup service to the client.

Brief Description of the Drawings

The accompanying drawings, which are incorporated in and constitute a part of the specification, illustrate presently preferred embodiments of the invention and, together with the general description given above and the detailed description of the preferred embodiments given below, serve to explain the principles of the invention.

Figure 1 depicts of a typical distributed network;

Figure 2 depicts a distributed system suitable for practicing methods and systems consistent with the present invention;

Figure 3a depicts a more detailed block diagram of a computer depicted in Figure 2;

Figure 3b depicts a more detailed block diagram of a lookup discovery computer depicted in Figure 2;

Figure 4 depicts a flowchart of the registration procedure in accordance with the present invention;

Figure 5 depicts a flowchart of the discovery process in accordance with the present invention; and

Figure 6 depicts a flowchart of the removal process in accordance with the present invention.

Detailed Description

In the following detailed description of the preferred embodiment, reference is made to the accompanying drawings that form a part thereof, and in which is shown by way of illustration a specific embodiment in which the invention may be practiced. This embodiment is described in sufficient detail to enable those skilled in the art to practice the invention and it is to be understood that other embodiments may be utilized and that structural changes may be made without departing from the scope of the present invention. The following detailed description is, therefore, not to be taken in a limited sense.

Overview of The Distributed System

Methods and systems consistent with the present invention operate in a distributed system ("the exemplary distributed system") with various components, including both hardware and software. The exemplary distributed system (1) allows users of the system to share services and resources over a network of many devices; (2) provides programmers with tools and programming patterns that allow development of robust, secured distributed systems; and (3) simplifies the task of administering the distributed system. Reference will be made to programs provided in the Java programming language, as described in James Gosling, Bill Joy, Guy Steele, "The Java Language Specification," Addison-Wesley, 1996, (hereinafter referred to as the "Java language specification"), incorporated herein by reference, which are processed in connection with an execution environment which is provided by a Java virtual machine. The Java virtual machine, in turn, is specified in the Lindholm and Yellin, "The Java Virtual Machine Specification," Addison-Wesley, 1996, incorporated herein by reference. As described in the Java language specification, programs in the Java programming language define "classes" and "interfaces." A "class" is a template from which an object can be created. It is used to specify the behavior and attributes common to all objects of the class. The mechanism by which new classes are defined from existing classes is "inheritance." It is the mechanism by which code reusability is facilitated. "Subclasses" of a class inherit operations of their parent or "superclass". An interface specifies the set of methods that must be implemented by any class that implements that interface (but does not provide any implementation of these methods itself). More specifically, the Java programming language provides for declarations wherein a user declares variables of type I, where I is an interface. Such variables, referred to as interface references, can store a reference to an object of any class that implements the interface I. Interface references can also be used to invoke any of the methods declared in the interface declaration. Importantly, the Java programming language provides for multiple inheritance among interfaces, i.e., an interface may extend multiple interfaces. Also, a class can implement multiple interfaces.

Fig. 2 depicts the exemplary distributed system 100 containing a server computer 102, a client computer 104, and a device 106 interconnected by a network 108. It will be appreciated that a client computer may also perform operations described herein as being performed by a server computer, and similarly a server computer may also perform operations described herein as being performed by a

client computer. Moreover, the device 106 may be any of a number of devices, such as a printer, fax machine, storage device, input device, computer, or other devices. The network 108 may be a local area network, wide area network, or the Internet. Although only two computers and one device are depicted as comprising the exemplary distributed system 100, one skilled in the art will appreciate that the exemplary distributed system 100 may include additional computers or devices.

Fig. 3a depicts the server computer 102 in greater detail to show a number of the software components of the exemplary distributed system 100. Computer 102 includes a memory 202, a secondary storage device 204, a central processing unit (CPU) 206, an input device 208, and a video display 210. The memory 202 includes a lookup service 212, a discovery server 214, and a Java runtime system 216. The Java runtime system 216 includes the Java remote method invocation system (RMI) 218 and a Java virtual machine (JVM) 220. RMI 218 facilitates the invocation of remote methods and JVM 220 acts like an abstract computing machine, receiving instructions from programs in the form of bytecodes and interpreting these bytecodes by dynamically converting them into a form for execution, such as object code and executing them. RMI 218 and JVM 220 are further described in the previously referenced application entitled "Dynamic Lookup Service in a Distributed System."

The lookup service 212 defines the services that are available for a particular Djinn. The lookup service 212 contains one object for each service within the Djinn, and each object contains various methods that facilitate access to the corresponding service. The lookup service 212 is described in greater detail in the previously referenced application entitled "Dynamic Lookup Service in a Distributed System."

The discovery server 214 detects when a new device is added to the exemplary distributed system 100, during a process known as boot and join (or discovery), and when such a new device is detected, the discovery server passes a reference to the lookup service 212 to the new device, so that the new device may register its services with the lookup service and become a member of the Djinn. After the device discovers the lookup service 212, the device may access any of the services registered with the lookup service 212. Furthermore, the device may also advertise its own services by registering with the lookup service 212. Once registered, the services provided by the device may be accessed through the lookup service 212, by all other entities that also discover lookup service

212. The process of boot and join is described in greater detail in the previously referenced application entitled "Apparatus and Method for providing Downloadable Code for Use in Communicating with a Device in a Distributed System."

FIG. 3b depicts the computer 104 in greater detail to show a number of the software components of the exemplary distributed system 100. Like server computer 102, computer 104 includes a memory 302, secondary storage device 304, CPU 306, input device 308, video display 310, and Java runtime system 316. Lookup discovery service 312 is comprised of a helper service class that any computer 104 or 102 can use to discover services, and register for event notification. A helper service is a programming component that can be used during construction of Jini services and/or clients. Helper services are typically remote. In other words, their methods execute on remote hosts. Consequently, they register with a lookup service and they are looked up by devices wishing to employ them.

Although systems and methods consistent with the present invention are described as operating in the exemplary distributed system and the Java programming environment, one skilled in the art will appreciate that the present invention can be practiced in other systems and other programming environments. Additionally, although aspects of the present invention are described as being stored in memory, one skilled in the art will appreciate that these aspects can also be stored on or read from other types of computer-readable media, such as secondary storage devices, like hard disks, floppy disks, or CD-ROM; a carrier wave from the Internet; or other forms of RAM or ROM either currently known or later developed. Sun, Sun Microsystems, the Sun Logo, Java, and Java-based trademarks are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

Lookup Discovery Service

The lookup discovery service may be implemented as a well-behaved Jini service, complying with all of the policies embodied in the Jini technology programming model. It defines the interface through which other Jini services and clients may request that discovery processing be performed on their behalf. When an entity registers with the lookup discovery service a proxy is returned. Through

the returned proxy the entity may manage the parameters reflected in the entity's registration with the lookup discovery service.

Table 1

```
package net.jini.discovery;
    {
        public interface LookupDiscoveryService
            public LookupDiscoveryRegistration register
                                        (String[ ] groups,
                                        LookupLocator[ ] locators,
                                        RemoteEventListener listener,
                                        MarshalledObject handback,
                                        long leaseDuration)
                                            throws RemoteException;
    }
```

As seen from the lookup discovery interface shown in Table 1, when requesting a registration with the lookup discovery service, the client indicates the lookup services it is interested in discovering by submitting two sets of objects. Each set may contain zero or more elements. One set comprises the names of the groups whose members are lookup services the client wishes to be discovered. The other set comprises LookupLocator objects, each corresponding to a specific lookup service the client wishes to be discovered.

For each successful registration, the lookup discovery service will manage both the set of group names and the set of locators submitted. For the purposes of this document, these sets will be referred to as the managed set of groups, and the managed set of locators respectively. The managed set of groups associated with a particular registration contains the names of the groups whose members consist of lookup services the client wishes to be discovered through multicast discovery. Similarly, the managed set of locators contains instances of LookupLocator, each corresponding to a specific lookup service the client wishes to be discovered through unicast discovery. The references to the lookup services that have been discovered will be maintained in a set referred to as the managed set of lookup services (or managed set of registrars). Note that when the general term managed set is used, it should be clear from the context whether groups, locators or registrars are

being discussed. Furthermore, it is understood that when the term "group discovery" or "locator discovery" is used, it should be taken to mean the employment of the multicast discovery protocol or the unicast discovery protocol, respectively to discover lookup services that correspond to members of the appropriate managed set.

Now referring to FIG. 4, in order to employ the lookup discovery service to perform discovery on its behalf, a client must first register (step 400). Clients register with the lookup discovery service by invoking the register method defined in the lookup discovery service interface (Table 1). As seen in Table 1, the register method is the only method specified by this interface. An invocation of the register method produces an object -- referred to as a registration object (or simply, a registration) -- that is mutable. That is, the object produced contains methods through which the object may be changed. Because of this, each invocation of the register method produces a new registration object. Thus, the register method is not idempotent. Referring to Table 1, it is shown that the register method takes the following arguments as input: a String array, none of whose elements may be null, comprising zero or more names of the groups to which the desired lookup services belong; an array of zero or more (non-null) LookupLocator objects, each corresponding to a specific lookup service the client wishes to be discovered; a RemoteEventListener object which specifies the entity that will receive events notifying the registration of the discovery of lookup services of interest; an instance of MarshalledObject which specifies an object that will be included in the notification event that the lookup discovery service sends to the registered listener; and a long value representing the amount of time (in milliseconds) for which the resources of the lookup discovery service are being requested. It is important to note that the entity that receives such an event notification does not have to be a client of the lookup discovery service, it may be any third party event handling service.

As shown in step 410 of FIG. 4, the register method returns an instance of the LookupDiscoveryRegistration interface. It is through this return object that the client interacts with the lookup discovery service. This interaction includes activities such as group and locator management, state retrieval, and requesting the re-discovery of previously discovered but unavailable ("discarded") lookup services. The semantics of the methods of the LookupDiscoveryRegistration interface are defined in the next section. A client's registration with the lookup discovery service is

persistent across restarts (crashes) of the lookup discovery service, until the lease expires or is canceled.

The groups argument shown in Table 1 takes a String array of non-null elements. A null reference (LookupDiscovery.ALL_GROUPS) may however be input to this argument. If null is input, the lookup discovery service will attempt to discover all lookup services located within the multicast radius of the host on which the lookup discovery service is running. On the other hand, if an empty array (LookupDiscovery.NO_GROUPS) is input, then group discovery (multicast) for that registration will not be performed until the client, through one of the registration's methods, populates the managed set of groups.

As with the groups argument, the locators argument takes a String array, none of whose elements may be null. If either the empty array or null is input to the locators argument, then locator discovery for the registration will not be performed until the client through one of the registration's methods, populates the managed set of locators.

Upon discovery of a lookup service, through either group discovery or locator discovery, the lookup discovery service as shown in step 420 will send an event, referred to as a discovery event, to the listener associated with the registration produced by the call to register. A valid parameter must be input to the listener argument of the register method. If null is input to this argument, then a NullPointerException will be thrown and the registration fails. The state information maintained by the lookup discovery service includes the set of group names, locators and listeners submitted by each client through each invocation of the register method, with duplicates eliminated. This state information contains no knowledge of the clients that register with the lookup discovery service. Thus, there is no requirement that a client identify itself during the registration process. For each registration created by the lookup discovery service, an event identifier will be generated that uniquely maps the registration to the set of groups and locators, as well as to the listener, submitted to the registration request. This event identifier is returned through the returned registration object, and is unique across all other active registrations with the lookup discovery service.

As shown in step 444, the lookup discovery service continuously searches for new lookup services. When it finds a new service, process flows to step 448, where the process determines whether the lookup service is "of interest" to one of its clients. If it is, the lookup discovery service

sends a RemoteDiscoveryEvent, containing the appropriate event identifier, to the listener corresponding to each such registration. If the lookup service is not of interest, process flows back to step 444 and the lookup discovery service goes back to performing discovery-related activities. Once an event signaling the discovery of a desired lookup service (by group or by locator) has been sent, no other discovery events for that lookup service will be sent to a registration's listener until the lookup service is discarded (through that registration) and then re-discovered. Note that a detailed definition of the process that occurs when the lookup discovery service discards a lookup service is presented later in this document. If, between the time a lookup service is discarded (through any registration) and the time it is re-discovered, a new registration is requested having parameters referencing that lookup service, then upon re-discovery of the lookup service an event will also be sent to that new registration's listener.

As shown in Table 1, when requesting a registration with the lookup discovery service, a client may also supply a reference to an object, wrapped in a MarshalledObject, referred to as a handback. When the lookup discovery service sends an event to a registration's listener, the event sent will also contain a reference to this handback object. This object is known to the remote event listener and should contain any information that is needed by the listener to identify the event and to react to the occurrence of that event. The semantics of the object input to the handback argument are left to each client to define (null may be input to this argument.)

When a client registers with the lookup discovery service, it is effectively requesting a lease on the resources provided by that service. The initial duration of the lease granted to a client by the lookup discovery service will be less than or equal to the requested duration reflected in the value input to the leaseDuration argument. That value must be positive, Lease.FOREVER or Lease.ANY. If any other value is input to this argument, an IllegalArgumentException will be thrown. The client may obtain a reference to the Lease object granted by the lookup discovery service through the registration object returned by the service.

As stated earlier with respect to registering a client with the lookup discovery service, when a client requests a registration with a lookup discovery service, an instance of the LookupDiscoveryRegistration interface is returned. It is through this interface that the client manages

the parameters reflected in the registration with the lookup discovery service. A typical LookupDiscoveryRegistration interface is shown in Table 2.

Table 2

```
package com.sun.jini.discovery;
public interface LookupDiscoveryRegistration
    {
        public EventRegistration getEventRegistration( );
        public Lease getLease( );
        public ServiceRegistrar[ ] getRegistrars( ) throws RemoteException;
        public String[ ] getGroups( ) throws RemoteException;
        public LookupLocator[ ] getLocators( ) throws RemoteException;
        public void addGroups(String[ ] groups) throws RemoteException;
        public void setGroups(String[ ]  groups) throws RemoteException;
        public void removeGroups(String[ ] groups) throws RemoteException;
        public void addLocators(LookupLocator[ ] locators)
                                                    throws RemoteException;
        public void setLocators(LookupLocator[ ] locators)
                                                    throws RemoteException;
        public void removeLocators(LookupLocator[ ] locators)
                                                    throws RemoteException;

        public void discard(ServiceRegistrar registrar) throws RemoteException;
    }
```

As with the lookup discovery service interface, this is not a remote interface. Each implementation of the lookup discovery service exports proxy objects that implement this interface local to the client, using an implementation-specific protocol to communicate with the actual remote server. All of the proxy methods must obey normal RMI remote interface semantics except where explicitly noted.

The discovery facility of the lookup discovery service, together with its event mechanism, make up the set of resources clients register to use. Because the resources of the lookup discovery service are leased, access is granted for only a limited period of time, unless there is an active expression of continuing interest on the part of the client. When a client, through the registration process, requests that a lookup discovery service perform discovery of a set of desired lookup services, the client is also registered with the service's event mechanism. Because of this, the lookup discovery service "bundles" both resources under a single lease. When that lease expires, both discovery processing and event notifications will cease with respect to the registration that resulted

from the client's request. To facilitate lease management and event handling, this interface defines methods which allow the client to retrieve its event registration information. Additional methods defined by this interface allow the client to retrieve references to the set of currently discovered lookup services, as well as to modify the managed sets of groups and locators. If the client's registration with the lookup discovery service has expired or been canceled, then any invocation of a remote method defined in this interface will result in a NoSuchObjectException. The methods defined by this interface are organized into a set of accessor methods, a set of group modification methods, a set of locator modification methods and the discard method. Through the accessor methods various elements of a registration's state can be retrieved. The modification methods provide a mechanism for changing the set of groups and locators to be discovered for the registration. Through the discard method a particular lookup service may be made eligible for re-discovery.

The getEventRegistration method returns an EventRegistration object that encapsulates the information needed by the client to identify a notification sent by the lookup discovery service to the registration's listener. This method is not remote and it takes no arguments.

The getLease method returns the Lease object that controls a client's registration with the lookup discovery service. It is through the object returned by this method that the client requests the renewal or cancellation of the registration with the lookup discovery service. This method is not remote, nor does it take any arguments.

Note that the object returned by the getEventRegistration method also provides a getLease method. The getEventRegistration and the getLease method defined by the LookupDiscoveryRegistration interface both return the same Lease object. The getLease method defined here is provided as a convenience to avoid the indirection associated with the getLease method on the EventRegistration object, as well as to avoid the overhead of making two method calls.

The getRegistrars method returns an array containing references to each lookup service that has already been discovered for the registration. Client's can employ this method to determine if an event containing a discovered lookup service was sent to the registration's listener, but was missed (i.e., was never received) by the listener.

The getGroups method returns an array comprising the group names from the registration's managed set. If the managed set of groups is empty, this method returns the empty array. If there is no managed set of groups associated with the registration, then null is returned.

The getLocators method returns an array comprising the LookupLocator objects from the registration's managed set. If the managed set of locators is empty; this method returns the empty array. If there is no managed set of locators associated with the registration, then null is returned.

With respect to a particular registration, the groups to be discovered may be modified using the methods described below. In each case, a set of groups is represented as a String array, none of whose elements may be null. The empty set is denoted by the empty array (LookupDiscovery.NO_GROUPS), and "no set" is indicated by null (LookupDiscovery.ALL_GROUPS). No set indicates that all lookup services within the multicast radius should be discovered. With respect to any set of the group names input to these methods, duplicated group names reflected in the input will be ignored.

The addGroups method adds a set of group names to the registration's managed set. This method takes one argument as input: a String array comprising the set of group names with which to augment the managed set. Elements in the input set that duplicate elements already in the managed set will be ignored. Once a new name has been added to the managed set, the lookup discovery service will attempt to discover all (as yet) undiscovered lookup services that are members of the group having that name. This method throws an UnsupportedOperationException if the registration has no current managed set of groups to augment. If null is input, this method throws a NullPointerException. If the empty array is input, then the registration's managed set of groups will not change.

The setGroups method replaces all of the group names in the registration's managed set with names from a new set. This method takes one argument as input: a String array comprising the set of new group names that will replace the set of names in the managed set. Once a new group name has been placed in the managed set, if there are lookup services belonging to that group that have already been discovered, no event will be sent to the registration's listener for those particular lookup services. Attempts to discover all (as yet) undiscovered lookup services belonging to that group will continue to be made for the registration. If null is input to setGroups, then the lookup discovery service will

attempt to discover all (as yet) undiscovered lookup services located within the multicast radius and, upon discovery of any lookup service, will send to the registration's listener an event signaling that discovery. If the empty array is input, then group discovery for the registration will cease.

The removeGroups method deletes a set of group names from the registration's managed set. This method takes one argument as input: a String array containing the set of group names to remove. This method throws an UnsupportedOperationException if the registration has no current managed set of groups from which to remove elements. If null is input, this method throws a NullPointerException. If the empty array is input, then the registration's managed set of groups will not change.

After a set of groups has been removed from the managed set because of an invocation of either setGroups or removeGroups, attempts to discover any lookup service that satisfies each of the following characteristics will cease to be made for the registration: (1) the lookup service is a member of one or more of the groups that was removed from the registration's managed set; (2) the lookup service is not a member of any group in the new managed set resulting from the invocation of setGroups or removeGroups; and (3) the lookup service does not correspond to any element in the registration's managed set of locators.

With respect to a particular registration, the locators to be discovered may be modified using the methods described below. In each case, a set of locators is represented as an array of LookupLocator objects, none of whose elements may be null. With respect to any set of locators input to these methods, duplicated locators (as determined by LookupLocator.equals) will be ignored. The addLocators method adds a set of LookupLocator objects to the registration's managed set. This method takes one argument as input: an array comprising the set of locators with which to augment the managed set. Elements in the input set that duplicate (using LookupLocator.equals) elements already in the managed set will be ignored. This method throws an UnsupportedOperationException if the registration has no managed set of locators to augment. If null is input to addLocators, a NullPointerException will be thrown. If the empty array is input, the registration's managed set of locators will not change.

The setLocators method replaces all of the locators in the registration's managed set with LookupLocator objects from a new set. This method takes one argument as input: an array

comprising the set of locators that will replace the locators in the managed set. If null is input to setLocators, a NullPointerException will be thrown. If the empty array is input, all locator discovery performed by the lookup discovery service, for the registration, will cease.

The removeLocators method deletes a set of LookupLocator objects from the registration's managed set. This method takes one argument as input: an array containing the set of locators to remove. This method throws an UnsupportedOperationException if the registration has no managed set of locators from which to remove elements. If null is input to removeLocators, a NullPointerException will be thrown. If the empty array is input, the registration's managed set of locators will not change.

Whenever a new locator is placed in the managed set as a result of an invocation of one of the methods above, the lookup discovery service will attempt unicast discovery of the lookup service associated with the new locator. It is important to note that discovery will not be attempted if the new locator equals any locator corresponding to the previously discovered lookup services (across all registrations); where equality is determined by the equals method of LookupLocator.

As shown in FIG. 5, when discovery is attempted (step 500), the discovery attempt will be repeated until one of the following events occurs: the lookup service is discovered (step 510); the client's lease expires (step 530); or the client explicitly removes the locator from the managed set (step 550). Upon discovery of the lookup service corresponding to the new locator or upon finding a match with a previously discovered lookup service, an event signaling a discovery will be sent to the registration's listener as shown in step 520.

After initial discovery of a lookup service, the lookup discovery service will continue to monitor the state of the multicast announcements from that lookup service (step 525). Depending on the state of those announcements, the lookup discovery service may send either a discovery event or an event referred to as a discard event (FIG. 6). A discovery event will be sent (step 585) in the event that the multicast announcements from the already discovered lookup service indicate that the lookup service is a member of a new group (step 560) and at least one registration has registered interest in one or more of the new groups (step 580).

A discard event, on the other hand, will be sent upon the occurrence of one of the following events:

- in response to a discard request resulting from an invocation of a registration's discard method;
- when a lookup service is removed from a registration's managed set in response to an invocation of either setLocators or removeLocators;
- when multicast announcement from an already-discovered lookup service are no longer being received (step 570);
- when multicast announcements from an already discovered lookup service indicate that the lookup service is no longer a member of one or more of the groups reflected in its previous multicast announcements, and there is no interest in any of the remaining groups, reflected in those announcements (step 590).

As shown in FIG. 6, when an existing locator is removed from the managed set as a result of an invocation of one of the methods above, the action taken by the lookup discovery service depends on whether the lookup service corresponding to that locator had been previously discovered for the registration. First, as shown in step 600, a request is received to remove a lookup service from the managed set. If the lookup service associated with the locator was previously discovered, processing flows to step 630 and the lookup service will be discarded. Next, in step 640, an event is sent to the registration notifying it that the lookup service has been discarded. If, in step 610, it is determined that the lookup service has yet to be discovered, processing flows to step 620 where the system discontinues attempts to perform further locator discovery of that particular lookup service. Even though a lookup service that has not yet been discovered, does not actually need to be discarded, the system must nonetheless discontinue attempts to discover the unavailable lookup service.

When the lookup discovery service removes an already-discovered lookup service from a registration's managed set(s), making the lookup service eligible for re-discovery, the lookup service is considered to be discarded. Whenever the lookup discovery service discards a lookup service, it will send an event to the registration to notify it that the lookup service has been discarded.

The discard method provides a mechanism for registered clients to inform the lookup discovery service of the existence of an unavailable lookup service; and to request that the lookup discovery service discard that lookup service. The discard method takes a single argument as input:

the proxy to the lookup service to discard. This method takes no action if the proxy input to this method equals none of the proxies reflected in the managed set.

In the event that a lookup service goes down or for some reason is unavailable, there will be no automatic notification of the occurrence of such an event. This means that for each of the registration's targeted lookup services, after a lookup service is initially discovered, the lookup discovery service will not attempt to discover that lookup service again (for that registration) until it is discarded. When a client determines that a discovered lookup service is no longer available, it is the responsibility of the client to inform the lookup discovery service -- through the invocation of the registration's discard method -- that the previously discovered lookup service is no longer available; and that attempts should be made to re-discover that lookup service for the registration. Typically, a client determines that a lookup service is unavailable when the client attempts to use the lookup service but receives a non-fatal exception or error (e.g., RemoteException) as a result of the attempt.

The lookup discovery service can act on behalf of numerous clients having access to the same lookup service. If that lookup service becomes unavailable, many of those clients may invoke discard between the time the lookup service becomes unavailable and the time it is re-discovered. Upon the first invocation of discard, the lookup discovery service will reinitiate discovery of the relevant lookup service for the registration of the client that made the invocation. For all other invocations made prior to re-discovery, the registrations through which the invocation is made are added to the list of registrations that will be notified when re-discovery of the lookup service does occur. That is, upon re-discovery of the lookup service, only those registrations through which this method is invoked will be notified.

While the present invention has been described in connection with an exemplary embodiment, it will be understood that many modifications will be readily apparent to those skilled in the art, and this application is intended to cover any adaptations or variations thereof. For example, different labels or definitions for the multicast packet may be used without departing from the scope of the invention. This invention should be limited only by the claims and equivalents thereof.